Using the 64k Paged Memory System

HOW IT WORKS:

Normally, the VIC-20 is only able to recognize up to 24k of expansion RAM (available from BASIC). Using the 64k Video Pak, however, provides 64k of expansion memory, but only if used properly! If the following procedures are used, all the available RAM on the Video Pak can be accessed from BASIC. The additional memory provided is not as easy to use as normal RAM, but with some care it can add tremendously to the convenience of your VIC system.

The address that the VIC assigns to expansion memory is in theory much like the address of houses on an imaginary street:

house#1: house#2 : house#3 : empty lot : house#4 :

If you wanted to build another house on this block, the only place you could build it would be on the empty lot, between houses 3 and 4. The operating system of the VIC is like this imaginary neighborhood; there is room for expansion but not very much. The way the VIC would increase the number of people that could live on this block would be to build only one more house:

house#1 : house#2 : house#3 : NEW HOUSE : house #4 :

There is no more room on this block and it would seem that no more poeple could be added here, but the 64k Video Pak has solved the problem in a different way. In effect, the Video Pak builds an apartment building in the same area as a house! Our imaginary block now looks like this:

```
                                    apartment#4
                                    apartment#3
                                    apartment#2
house#1  :  house#2  :  house#3  :  apartment#1  :  house#4
```

Building this way allows us to build enough space for as many people as we want, limited only by the maximum height of the buildings. There are some drawbacks, if you live in a house, you can walk in any time, but in an apartment building you may have to wait for the elevator (or even worse, take the stairs!). It is also more difficult to move in and out, etc..

In reality,the VIC memory map looks like this:

```
                    :                              :added memory :
                    :                              : goes here   :
                    :                              :             :
Working Storage:Expansion&Screen RAM:Expansion RAM:Roms & I/O
  0-1023            :     1024-8191      :  8192-32767 :32768-65535
```

The easiest way to fill the memory map is to add a 24k expansion board:

```
                    :                              :             :
Working Storage:Expansion&Screen RAM:Expansion RAM:Roms & I/O
  0-1023            :     1024-8191      :  8192-32767 :32768-65535
```

The 64k Video Pak adds memory in one of two ways, in 24k "page" mode:

```
                    :                              :    24k #2   :
Working Storage:Expansion&Screen RAM:    24k #1    :Roms & I/O
  0-1023            :     1024-8191      :  8192-32767 :32768-65535
```

Or in the 16k "page mode":

```
                                           :    16k #4   :
                                           :    16k #3   :
                    :                      :    16k #2   :
Working Storage:Expansion&Screen RAM:    16k #1    :Roms & I/O
  0-1023            :     1024-8191      :  8192-32767 :32768-65535
```

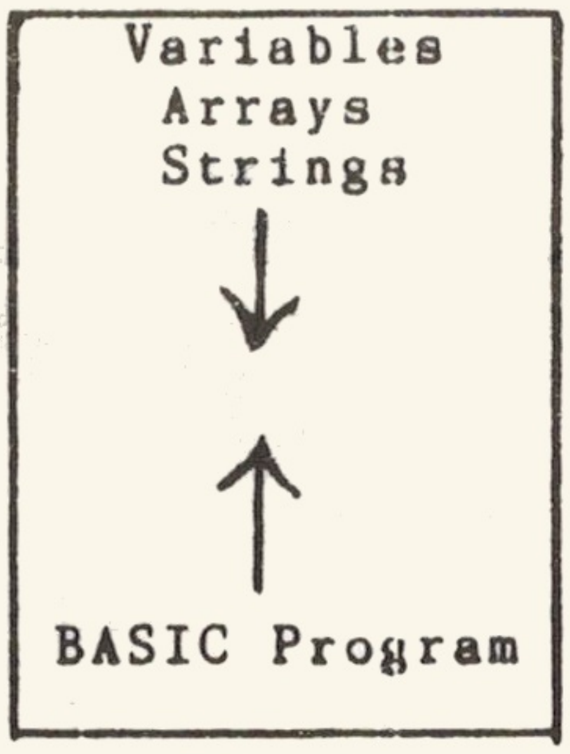
HOW TO WORK IT:

In order to use the 64k "paged memory" system, you will need to enter and run the following routine. This must be done before loading a program that uses 64k of RAM.

The default memory mode for the 64k Video Pak is two 24k memory pages. The same method required for the 16k pages applies for the 24k pages. Please note however, that in the 24k mode only 48k total of expansion RAM is available. Quite often not the entire 64k will be utilized and two 24k pages may prove sufficient. The simplicity of only having two memory pages to keep track of, might outway the disadvantage of 16k less of memory. In any case the decision to use either page mode should be made at the beginning, as it is nearly impossible to change the page mode without losing memory contents.

The first step in understanding the purpose of "paged memory" is to look at how the VIC-20 stores BASIC programs and variables in memory.

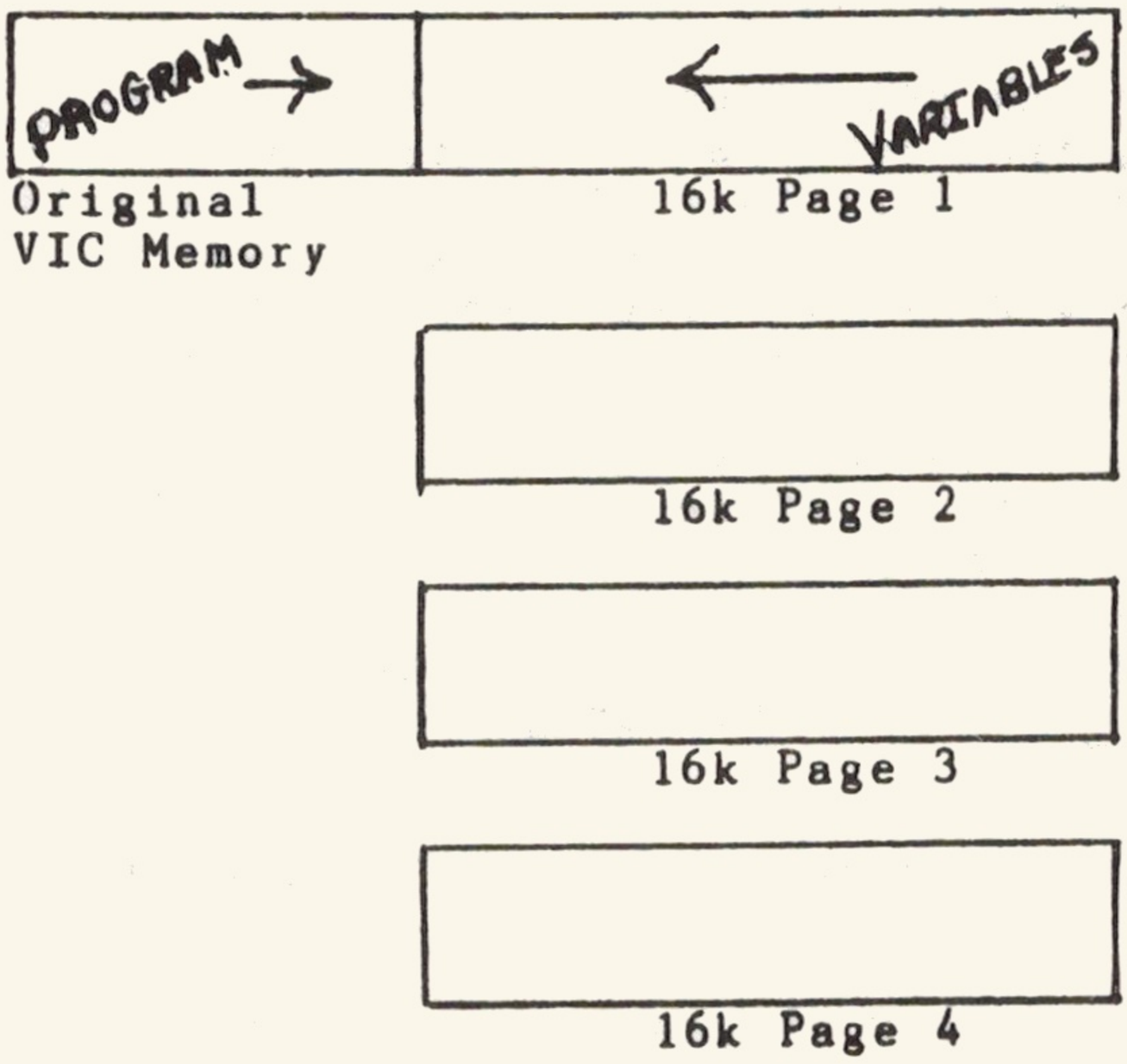
For example, lets' say you were using location 4096 - 7680 for your BASIC memory.



Your program begins at location 4096 and is stored upward towards the end of memory.

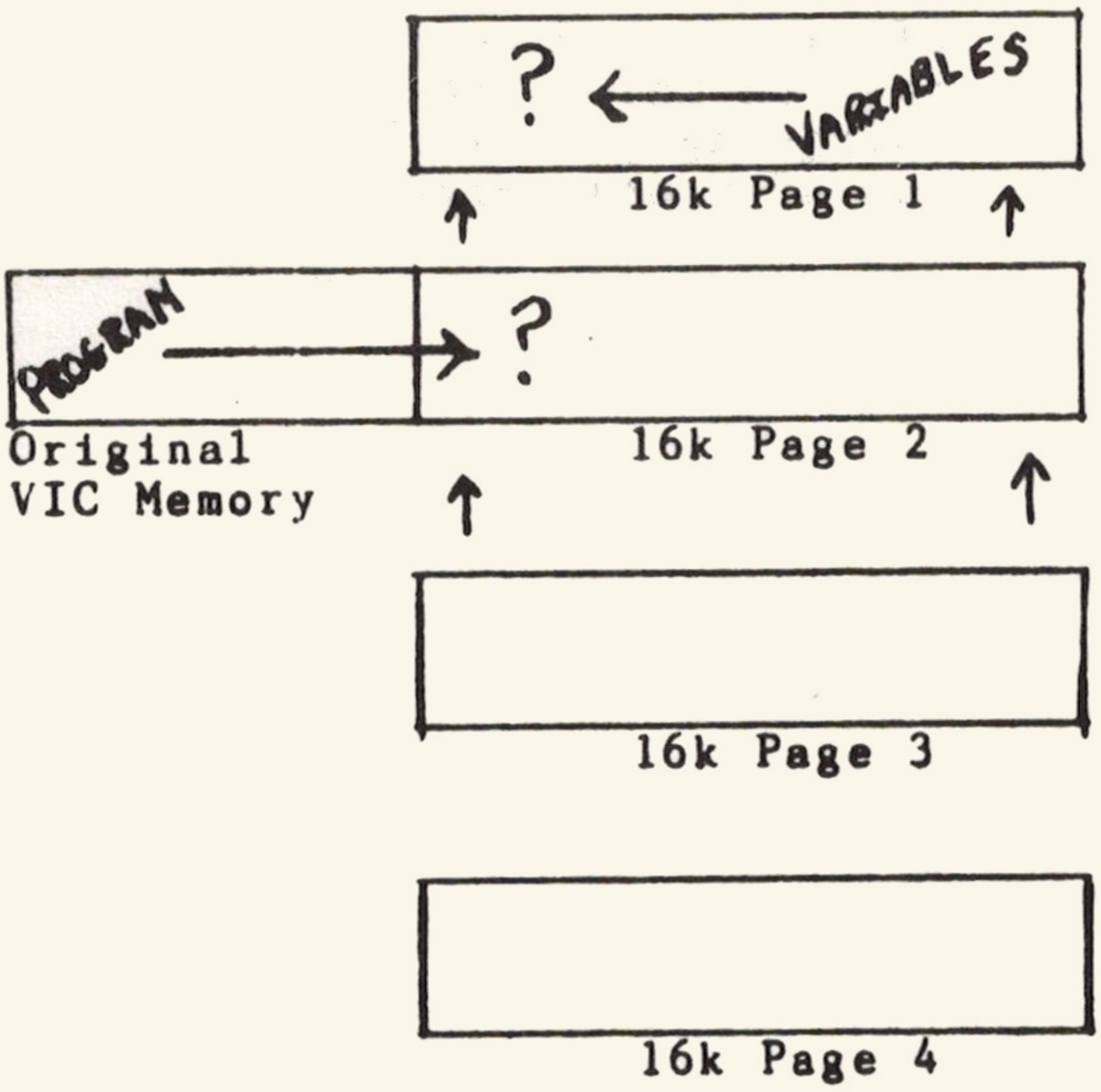
Variables, arrays and strings are stored beginning at the end of memory and work downward. If the upgoing pointer meets the downgoing pointer you have an out-of-memory-error.

This is the reason you must change your BASIC memory locations when using "paged memory". If you didn't you would lose part of your program. For example:

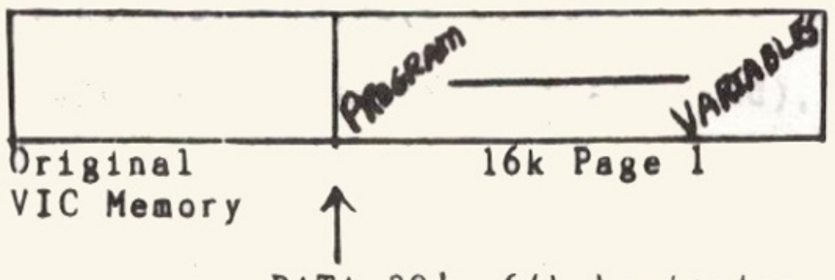


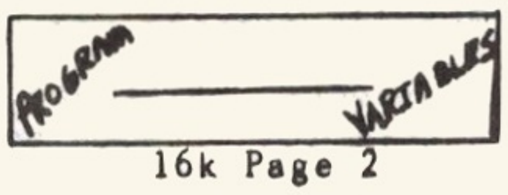
64k provided by DATA 20's Video Pak.

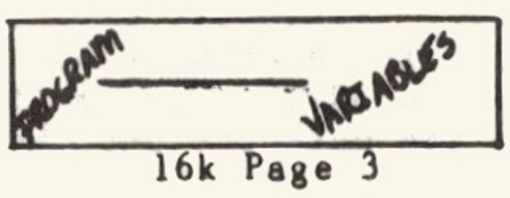
As you filled the first 16k of memory and went to the second "page", you would lose your variables and the program would crash!

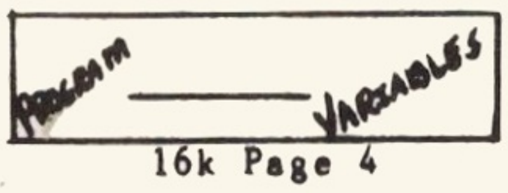


You can avoid this problem by making sure your 64k of BASIC memory begins at the bottom of expansion RAM.









You do this by entering the following routine:

```
10 POKE52,32:POKE56,32:POKE225,PEEK(225)OR2:D=PEEK(225):
   POKE39932,DOR2
20 FORJ=0TO3:POKE39932,(DAND51)+J*4:FORI=0TO2:POKE8192+I,
   0:NEXTI:NEXTJ
30 POKE642,32:POKE644,96:POKE39932,D:SYS58232
```

The first part of line 10 sets the top of BASIC below the bottom of expansion RAM. The purpose of this is to utilize the VIC-20's original memory to run the formatting program listed above. This BASIC setting is only temporary, the pointers will be reset after the formatting program has been run.

The next part of line 10 resets the memory to the "16k page" mode. (The Video Pak automatically goes to the "24k page mode" on power up.) In order to alter upper/lower case, "page" modes, or page numbers you must POKE into memory location 39932. Unfortunatley due to the VIC-20 memory configuration you cannot PEEK into this location. DATA 20 has duplicated this location and called it location 225. Making location 225 a "scratchpad" of location 39932. It is this location that is crucial on power-up to see what the user has selected.

Line 20 in the program pokes three zeros into the beginning of each 16k "page". This signifys these pages are blank and ready for programming.

Now that you have set up your four "pages" of 16k how do you utilize them, when writing a program over 16k? You do this by linking them together, which can be done with the following routines:

```
10 GOTO 100
20 POKE39932,(PEEK(225)AND51)+P*4
100 PRINT"THIS ";
1000 P=1:GOTO 20
```

By replacing line 100 with your program this small routine will utilize your first "page" of 16k. Then when executed it will automatically send you to the next 16k"page". Where you will enter the following routine:

```
10 GOTO 100
20 POKE39932,(PEEK(225)AND51)+P*4
100 PRINT"USES ";
1000 P=2:GOTO 20
```

Again by replacing line 100 with your program this small routine will utilize your second "page" of 16k and send you to the third "page". Where you will enter the following routine:

```
10 GOTO 100
20 POKE39932,(PEEK(225)AND51)+P*4
100 PRINT"FOUR ";
1000 P=3:GOTO 20
```

The forth routine is as follows:

```
10 GOTO 100
20 POKE39932,(PEEK(225)AND51)+P*4
100 PRINT"PAGES"
1000 P=0:GOTO 20
```

When typing in these 4 routines be sure to type the lines in exactly as shown. Otherwise the locations will be confused and you will get a syntax error.